History:

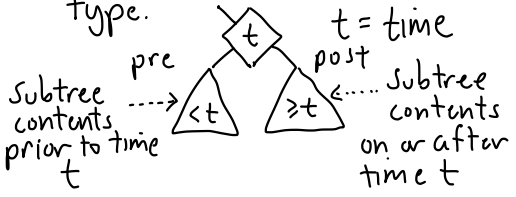
- Driscoll, Sarnak, Sleator, Tarjan (1986) - First serious theoretical analysis (Splay trees)
  → Applied to **geometric search** (time is coordinate)

Approaches:

Copy-on-write: Whenever you modify, make a copy
→ very inefficient!

Change-log: Make a list of recent updates
→ slow to process

Fat nodes / Node copying: When changes occur, save just modified portions
→ **Temporal node:** New node type.



Persistent Data Structures:

- Preserves **prior states** of data structure
- Allows **searches in history**
  → "Was Jane Smith enrolled in UMD in Spring 2016?"
- **Full/Partial Persistence:**
  - change can be made to any/current version

Persistent Search Trees I

Case study: PBJ Tree

Persistent Weight-Balanced Jackhammer trees

→ A **partially persistent** BJ tree
→ Uses **rebuilding** to balance subtrees
→ Allows **weighted entries**

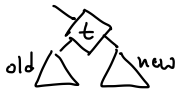
Example: Rebuild subtree T_1 at time t. Let T_2 be new subtree:

BJ Tree:

PBJ Tree:

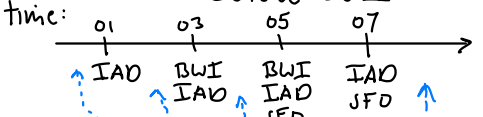
Approach: Whenever a modification made - save old contents & use temporal node to distinguish

Change at t:



Example:

Time:	Command:
01	insert IAD
03	insert BWI
05	insert SFO
07	delete BWI



find IAD at 00: not found
find IAD at 02: found
get-min at 04: BWI
get-min at 08: IAD

Persistent Search Trees II

PBJ Tree (Private) Data:

- final float ALPHA, BETA
  - ↳ same as BJ Tree
- Node root — root → init: null
- int firstInsertTime
- int lastInsertTime

→ init: -1

PBJ Tree: Tech Specs

Node structure:

- Node: weight + maxWt (float)
  - ↳ TempNode: time (int)
    - pre, post (Node)
- IntNode: key, left, right
- ExtNode: key, value

Few More Things:

- Before insert, check that t > lastInsertTime
  - → else Error!
- Set lastInsertTime ← t
- Note: Partial persistence
- Rebalance → Next

Insertion (without rebalancing)

Helper: Node insert(x, v, w, t)

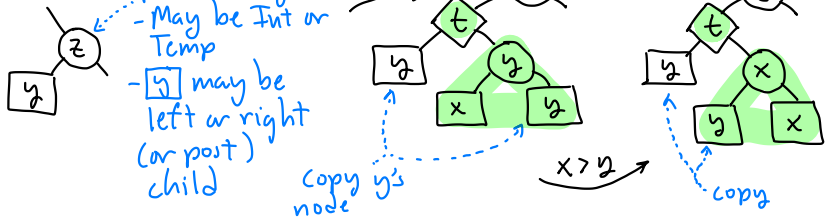
→ insert (x, v) of wt w at t

- If root == null: (first insertion)
  - firstInsertTime ← t
  - root = new ExtNode (x, v, w)
- else
  - search for x in tree
  - reach ExtNode y
    - if x == y ⇒ Error: Duplicate key
    - else:

Int: left ← left.insert(...)
right ← right.insert(...)
Temp: post ← post.insert(...)

→ Update node weight + maxWt (later)

Parent of y
- May be Int or Temp
- y may be left or right (or post) child

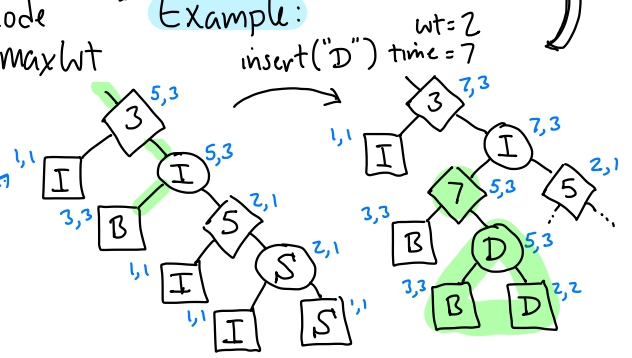


Updating Weights?

IntNode: wt ← left.wt + right.wt
maxWt ← max(left.maxWt, right...)

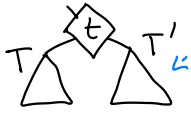
TempNode: wt ← post.wt ← only post!
maxWt ← post.maxWt

Example:



Internal Node Rebalance:

- Test α, β condition (same as BST tree)
- If unbalanced, compile list A of extern nodes for **current tree** ..... For temporal recurse only on post side
- T = original tree
- T' = buildTree(A)
- return:



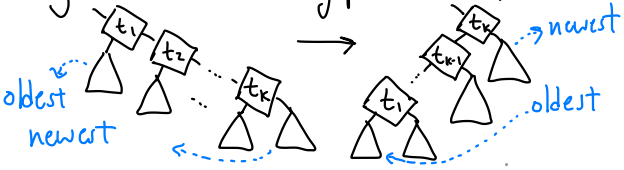
The new tree has no temporal nodes

Temporal Node Rebalance:

- Recurse on post side:
  post = post.rebalance(x,t)
- On return:
  if (post child is temporal)...
  - perform left rotation
  - update weights

  return root of subtree

Why? Don't like long post chains



Rebalancing after insertion:

- Starting at root, retrace search path
- Recursive helper:
  Node rebalance(x,t)
  → **Internal**: Apply to left/right based on x ↔ key
  → **Temporal**: Apply to post only.

Persistent Search Trees III

find(x,t), findUp(x,t), getMin(t)...

- same as before but for temporal nodes visit pre/post based on t
- check whether t < firstUpdateTime → if so → null

getPreorderList(t)/getFullPreorderList()

- First gets preorder list at time t (no temporal nodes!)
- Second gets full preorder list (all nodes)

Delete/Clear: Not implemented/Not required! (A bit messy)

Example: